Computer Vision

Computer Vision problems:

- image classification
- Object detection
- Neural style Transfer

Deep learning on large images.

Small image: $64 \times 64 \times 3 = 12288$ feasible input for Deep neural network.

Actual images in daily life: $1000 \sim \times 1000 \sim \times 3 = 3$ million (this is a retutively small image in actuall life).

input size: $(3M, m)$

layer 1: $(n_1, \boxed{3M})$

This is getting unpractical. (too many weights in the network).

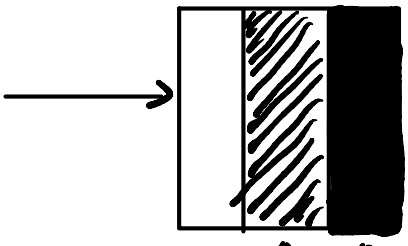
* solution: convolution.

Edge Detection

vertical edge detection:

filter: (kernel)

1	0	-1
1	0	-1
1	0	-1



This finds shapes in images where left is bright and right is dark (an edge)

Instead of specifying the filter explicitly, we can let computer learn to detect edges

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

padding :

Applying a filter on an image has 2 down effects :

- Shrinking output (3x3 filter on 6x6 image, resulting in a 4x4 image)
- * most importantly : throw away information from edge : edge pixels are used less frequently while applying filter.

padding adds a layer of blank (zero) pixels outside of the image, solving above 2 problems.

$$\underset{\text{image}}{n \times n} \longrightarrow n-f+2p+1 \quad \times \quad n-f+2p+1$$

where f = filter length

p = padding

valid and same convolutions :

valid : Size of image shrinks after convolution

same : keep output size the same as input size by padding.

Strided convolutions :

filters don't have to move across images pixel by pixel. Instead, it can be move by any size for each step :

stride = 2 : move 2 pixels a time.

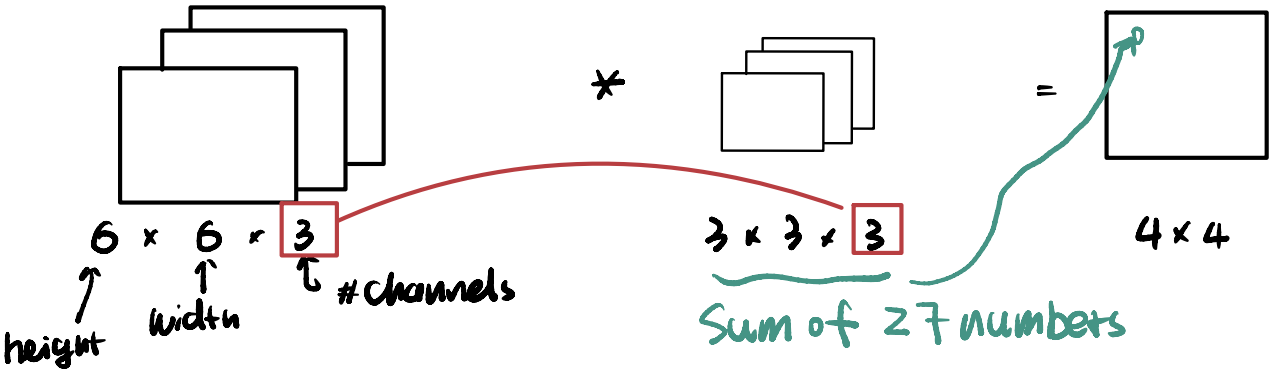
$$n \times n \longrightarrow \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \quad \times \quad ''$$

Strides, $f \times f$ filter, padding p.

One layer of convolutional network

convolutions over volumes :

common example : convolutions on RGB images



Example :

R

1	0	-1
1	0	-1
1	0	-1

G

0	0	0
0	0	0
0	0	0

B

0	0	0
0	0	0
0	0	0

Extract vertical lines in Red channel

R

1	0	-1
1	0	-1
1	0	-1

G

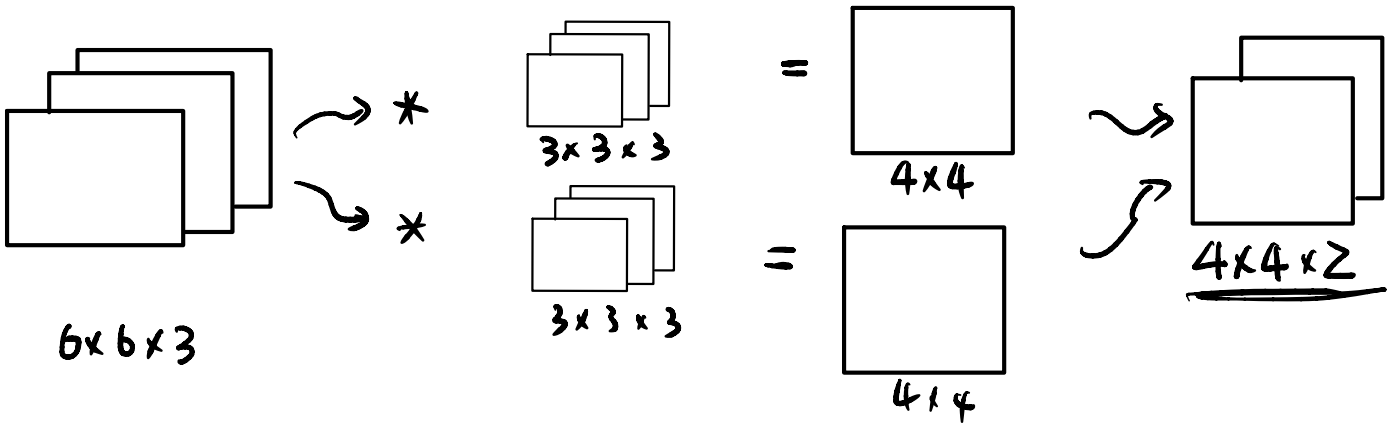
1	0	-1
1	0	-1
1	0	-1

B

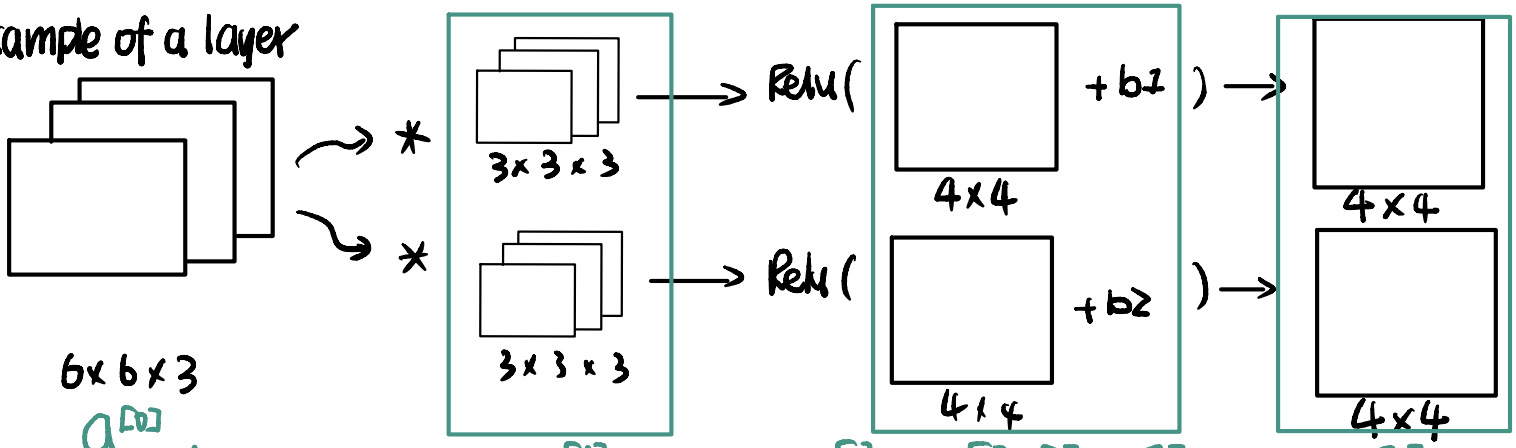
1	0	-1
1	0	-1
1	0	-1

Extract vertical features in all channels

Multiple filters :



Example of a layer



$6 \times 6 \times 3$

$a^{[0]}$

$W^{[1]}$

$Z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$

$a^{[1]}$

* Similar idea to standard neural network (non-convoluted NN)

Analysis: if you have 10 filters that are 3×3 in one layer of NN,

$$\#\text{Parameters} = (\underbrace{3 \times 3 \times 3}_{w} + \underbrace{1}_{b}) \times 10 = 280$$

Way less weights than non-convoluted neural network.

Summary of notation:

$f^{[l]}$: filter size

$p^{[l]}$: padding

$s^{[l]}$: stride

$n_c^{[l]}$: number of filters

input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$

output: $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

$A^{[l]}: \underbrace{m}_{\text{1st index}} \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

Each filter is: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

Activations: $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

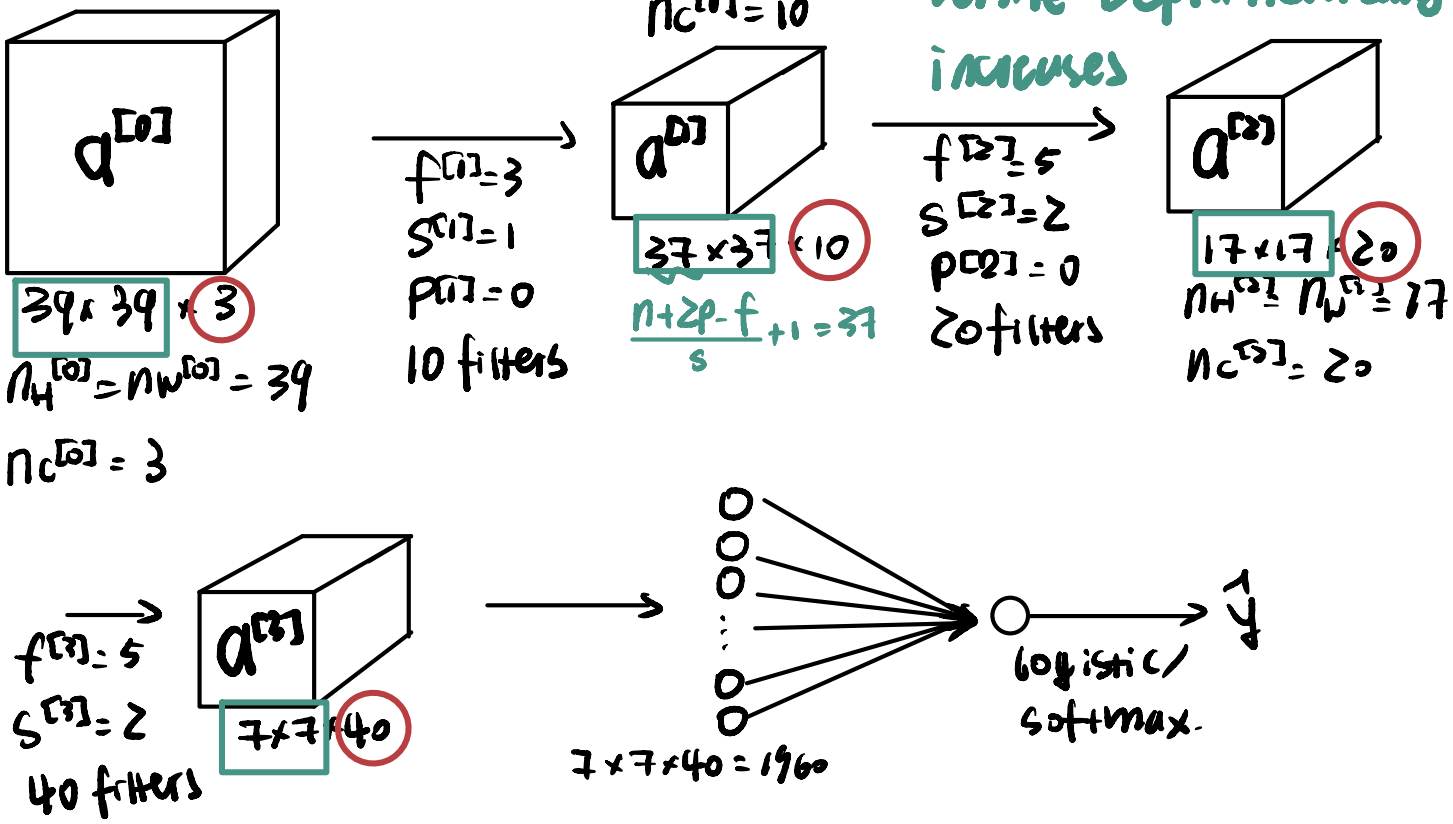
$$\underbrace{n_{H/W}^{[l]}}_{} = \frac{n_{H/W}^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1$$

Weights: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times \underbrace{n_c^{[l]}}_{\#\text{ filters at this layer.}}$

bias: $n_c^{[l]} - (1, 1, 1, n_c^{[l]})$

Example ConvNet

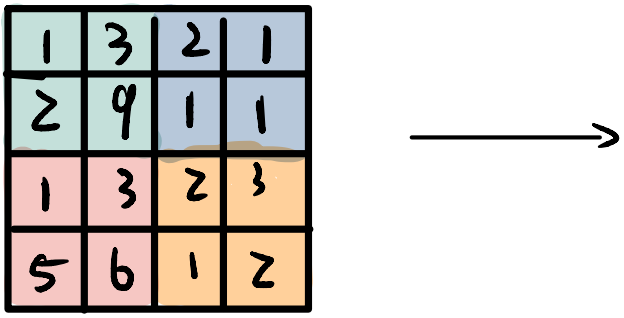
Notice that convolution size get trim down, while depth normally increases



* Types of layer in convolutional network.
  - Convolution (Conv)
  - Pooling (Pool)
  - Fully connected (FC)

Pooling Layers

max pooling



e.g. $9 = \max(1, 2, 3, 9)$

Same output size as:

$f = 2$
$s = 2$

hyperparameters

There are other types of pools:

e.g. average pooling (not used as frequently).

Summary:

Hyperparameter:

f: filter size

s: stride

Max or average pooling

Common values:

f = 2 s = 2

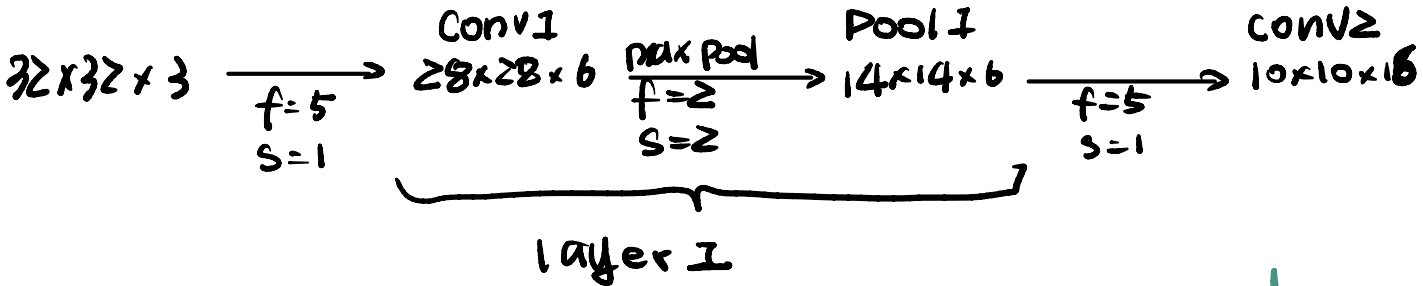
f = 3 s = 2

* No parameters to learn.

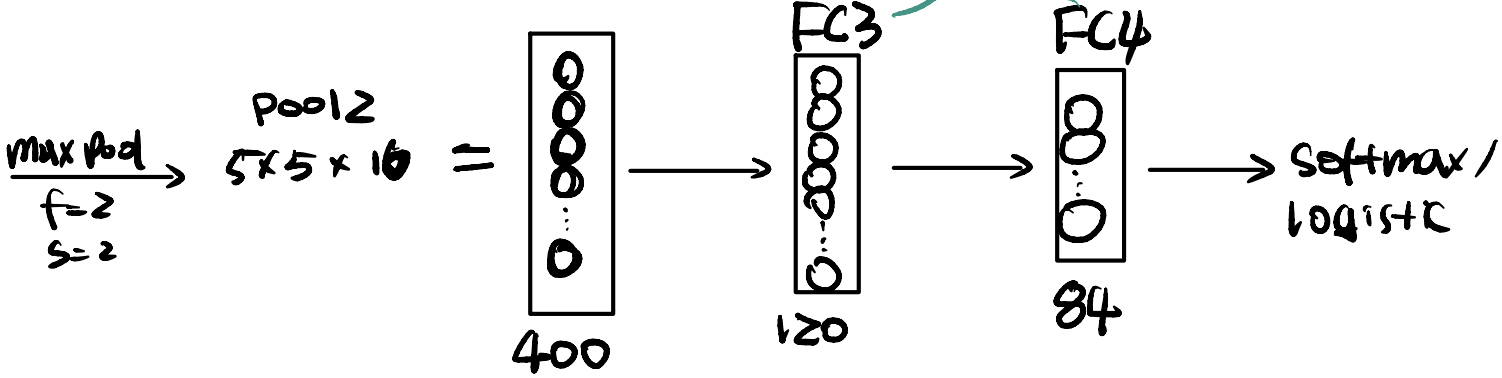
CNN Example

P.S. Conv1 and Pool1 are considered together as layer 1, since Pool1 has no weights. Pooling not considered as independent layer

Example: LeNet-5



Fully connected layer



Common pattern: n_H, n_W increase

n_C decrease.

Why Convolutions?

- parameter sharing:

  e.g. a horizontal line detector can work well cross the whole image with only a small number of weights.

- Sparsity of connections:

  in each layer, each output value depends only on a small number of inputs.

  Avoid variance given the small amount of weights CNN has.

Training set: $(x^{(1)}, y^{(1)}) \cdots (x^{(m)}, y^{(m)})$

Cost: $J = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$

Use gradient descent to optimize parameters to reduce J